Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота

з дисципліни “Методи синтезу віртуальної реальності”  
на тему “Spatial audio”

Варіант 16

Виконав студент 5-го курсу

групи ТР-31 мп  
Мітлицький Сергій Юрійович

Київ 2024

**Завдання**

Імплементувати просторове аудіо за допомогою WebAudio HTML5 API, використовуючи код з практичного завдання 2.

Імплементувати обертання джерела звуку відносно геометричного центру поверхні з використанням дотичного інтерфейсу (смартфону, що використовує сенсори для обробки інформації, щодо положення в просторі). На відміну від попередніх робіт поверхня залишається нерухомою - рухається джерело звуку. Відтворити улюблену пісню в форматі mp3/ogg, з можливістю контролювати місцезнаходження джерела звуку в просторі користувачем.

Візуалізувати джерело звуку за допомогою сфери.

Додати звуковий фільтр за варіантом (використовуючи інтерфейс BiquadFilterNode). Додати інтерфейс користувача, який би вмикав/вимикав фільтр. Встановити параметри фільтру відповідно до вподобань.

Підготувати звіт в цифровому вигляді, який би містив необхідні частини, що сповна описують поставлені задачі та виконану роботу.

**Теоретичні відомості**

**WebAudio API**

Аудіо в Інтернеті до цього моменту було досить примітивним і донедавна доводилося доставляти через плагіни, такі як Flash і QuickTime. Введення елемента аудіо в HTML5 є дуже важливим, оскільки дозволяє відтворювати базове потокове аудіо. Але він недостатньо потужний, щоб працювати зі складнішими аудіододатками. Для складних веб-ігор або інтерактивних програм потрібне інше рішення. Ціль цієї специфікації полягає в тому, щоб включити можливості сучасних ігрових звукових движків, а також деякі завдання мікшування, обробки та фільтрації, які є в сучасних настільних програмах створення звуку.

Інтерфейси програмного інтерфейсу були розроблені з урахуванням широкого спектру випадків використання [webaudio-usecases]. В ідеалі він повинен мати можливість підтримувати будь-який варіант використання, який можна розумно реалізувати за допомогою оптимізованого механізму C++, який керується через сценарій і запускається в браузері. Тим не менш, сучасне аудіопрограмне забезпечення для настільних комп’ютерів може мати дуже розширені можливості, деякі з яких було б важко або неможливо створити за допомогою цієї системи. Apple Logic Audio є однією з таких програм, яка підтримує зовнішні MIDI-контролери, довільні плагіни аудіоефектів і синтезаторів, високооптимізоване читання/запис аудіофайлів прямо на диск, тісно інтегроване розтягування часу тощо. Тим не менш, запропонована система буде цілком здатна підтримувати широкий спектр досить складних ігор та інтерактивних програм, включаючи музичні. І це може бути гарним доповненням до розширеніших графічних функцій, які пропонує WebGL. API розроблено таким чином, щоб пізніше можна було додати розширені можливості.

**BiquadFilterNode**

BiquadFilterNode — це процесор AudioNode, який реалізує дуже поширені фільтри нижчого порядку.

Фільтри нижчого порядку є будівельними блоками базових регуляторів тембру (баси, середні та високі частоти), графічних еквалайзерів і більш розширених фільтрів. Кілька фільтрів BiquadFilterNode можна комбінувати для створення більш складних фільтрів. Параметри фільтра, такі як частота, можна змінювати з часом для розгортки фільтра тощо. Кожен BiquadFilterNode можна налаштувати як один із кількох загальних типів фільтрів, як показано в IDL нижче. Тип фільтра за замовчуванням – «НЧ».

Фільтр низьких частот пропускає частоти нижче граничної частоти та послаблює частоти вище граничної. Він реалізує стандартний резонансний фільтр низьких частот другого порядку зі спадом 12 дБ/октаву.

Фільтр високих частот є протилежністю фільтру низьких частот. Частоти вище граничної частоти пропускаються, але частоти нижче граничної послаблюються. Він реалізує стандартний резонансний фільтр високих частот другого порядку зі спадом 12 дБ/октаву.

Смуговий фільтр пропускає діапазон частот і послаблює частоти нижче та вище цього діапазону частот. Він реалізує смуговий фільтр другого порядку.

Фільтр Lowshelf пропускає всі частоти, але додає підсилення (або ослаблення) нижніх частот. Він реалізує фільтр низького рівня другого порядку.

Фільтр Highshelf є протилежністю фільтру Lowshelf і пропускає всі частоти, але додає посилення до вищих частот. Він реалізує фільтр високої полиці другого порядку

Піковий фільтр пропускає всі частоти, але додає підсилення (або ослаблення) до діапазону частот.

Режекторний фільтр (також відомий як смуговий або смуговий фільтр) є протилежністю смуговому фільтру. Він дозволяє пропускати всі частоти, крім набору частот.

**Деталі впровадження**

Реалізував обертання джерела звуку навколо геометричного центру ділянки поверхні круговим способом протягом певного часу (цього разу поверхня залишається нерухомою, а джерело звуку рухається). Джерело звуку візуалізовано у вигляді сферичної геометрії

В подальшому цей функціонал дозволить керувати положенням джерела звуку просторового аудіо всередині програми.

Першим кроком для імплементації просторового аудіо є створення HTML-елементу <audio>, який містить інформацію про джерело аудіо-доріжки в моєму випадку це “filename” в форматі mp3/ogg. Цей елемент керування дозволить зупиняти та продовжувати відтворення аудіо-доріжки. Після цього за допомогою JavaScript, а саме WebAudio API було створено обʼєкт аудіоконтексту (AudioContext), для якого було створено та підʼєднано 3 основних обʼєкти:

* джерело звуку (MediaElementSource)
* обʼєкт обробки просторового аудіо (Panner)
* звуковий фільтр (BiquadFilter)

Згідно з варіантом було обрано фільтр низьких частот. Наступним кроком було встановлено параметри обраного фільтра. Взагалі, BiquadFilter має 3 параметри для налаштування:

* frequency (частота)
* Q (ширина смуги)
* gain (підсилення)

Особливість фільтру низьких частот в тому, що frequency - встановлює частоту зрізу Q - контролює, наскільки піковим буде відгук на частоті зрізу. Велике значення робить відповідь більш піковою. gain - не використовується в цьому типі фільтра. Джерело звуку, обробка якого в просторі здійснюється обʼєктом класу Panner зображено у WebGL контексті у вигляді сфери, щоб може переміщуватись - відповідно змінюючи параметри обʼєкту Panner, щоб при прослуховуванні аудіо зі стерео звуком був ефект переміщення джерела звуку відповідно до місцезнаходження сфери в системі координат.

**Інструкція для роботи з програмою**

Для налаштування стерео зображення використовуються 4 слайдери, в який можна налаштувати наступні параметри:

* Convergence (збіжність)
* Eye separation (відстань між очима)
* Field of view (поле зору)
* Near clipping distance (відстань ближньої площини, що відсікає зображену на екрані геометрію)

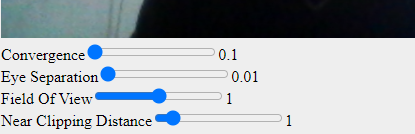


Рисунок 4.1 - скрішот слайдерів

Згідно до встановлених параметрів стерео зображення фігури виглядає дещо інакше

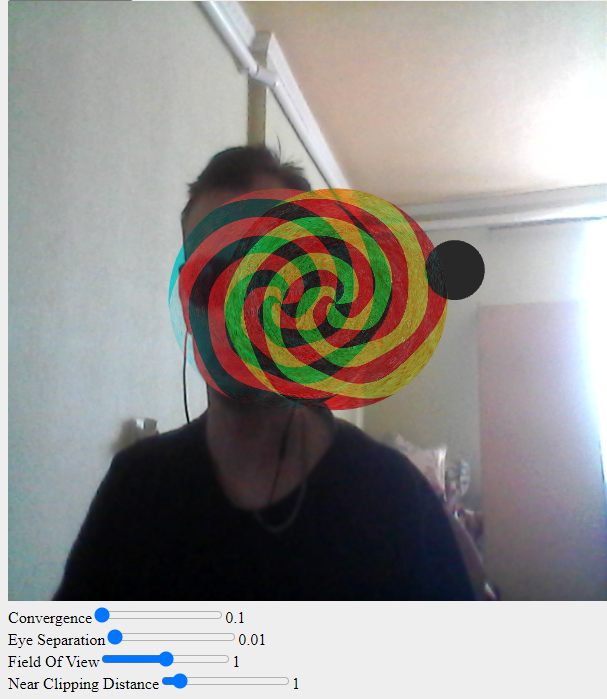


Рисунок 4.2 - скріншот фігури з початковими налаштуваннями

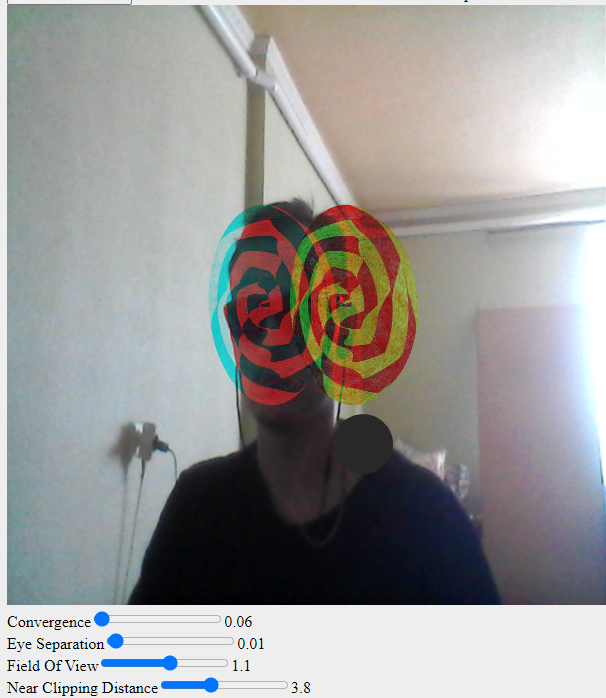


Рисунок 4.3 - скріншот фігури зі зміненими налаштуваннями

Вище згаданий HTML-елемент audio дозволяє керувати аудіо-доріжкою, а саме зупиняти та продовжувати відтворення, перемотувати на потрібний час, керувати гучністю.

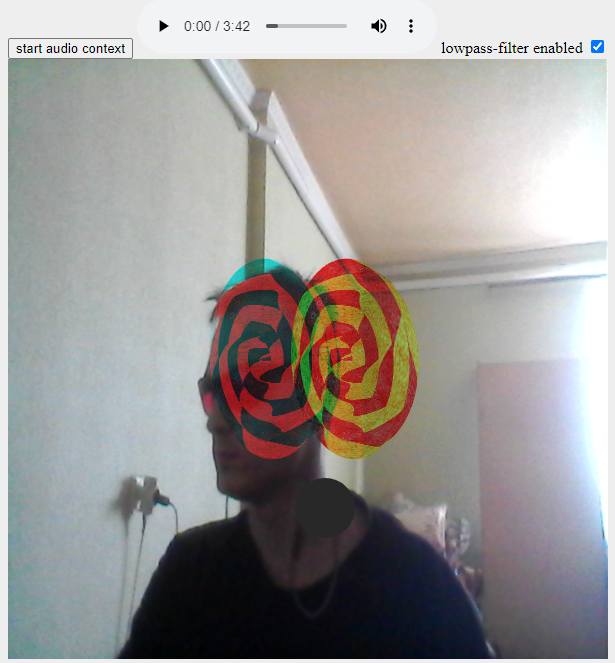


Рисунок 4.4 - скріншот з видимим елементом інтерфейса WEB-програвача

Також на попередньому рисункові можна побачити “Чекбокс lowpass-filter”, який дозволяє керувати станом звукового фільтру: вмикати та вимикати його. В залежності від стану фільтру можна помітити зміну звучання.

**Код програми**

//webgl context.

let gl;

//surface model

let surface;

//shader program

let shProgram;

// simple rotator

let spaceball;

let ball;

let texball = [0, 0]

let frustum;

// Constructor

function Model(name) {

    this.name = name;

    this.iVertexBuffer = gl.createBuffer();

    this.iNormalBuffer = gl.createBuffer();

    this.iTextureBuffer = gl.createBuffer();

    this.count = 0;

    this.BufferData = function (vertices, normals, textures) {

        gl.bindBuffer(gl.ARRAY\_BUFFER, this.iVertexBuffer);

        gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(vertices), gl.STREAM\_DRAW);

        gl.bindBuffer(gl.ARRAY\_BUFFER, this.iNormalBuffer);

        gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(normals), gl.STREAM\_DRAW);

        gl.bindBuffer(gl.ARRAY\_BUFFER, this.iTextureBuffer);

        gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(textures), gl.STREAM\_DRAW);

        this.count = vertices.length / 3;

    }

    this.Draw = function () {

        gl.bindBuffer(gl.ARRAY\_BUFFER, this.iVertexBuffer);

        gl.vertexAttribPointer(shProgram.iAttribVertex, 3, gl.FLOAT, false, 0, 0);

        gl.enableVertexAttribArray(shProgram.iAttribVertex);

        gl.bindBuffer(gl.ARRAY\_BUFFER, this.iNormalBuffer);

        gl.vertexAttribPointer(shProgram.iAttribNormal, 3, gl.FLOAT, false, 0, 0);

        gl.enableVertexAttribArray(shProgram.iAttribNormal);

        gl.bindBuffer(gl.ARRAY\_BUFFER, this.iTextureBuffer);

        gl.vertexAttribPointer(shProgram.iAttribTexture, 2, gl.FLOAT, false, 0, 0);

        gl.enableVertexAttribArray(shProgram.iAttribTexture);

        gl.drawArrays(gl.TRIANGLES, 0, this.count);

    }

}

// Constructor

function ShaderProgram(name, program) {

    this.name = name;

    this.prog = program;

    // Location of the attribute variable in the shader program.

    this.iAttribVertex = -1;

    // Location of the uniform specifying a color for the primitive.

    this.iColor = -1;

    // Location of the uniform matrix representing the combined transformation.

    this.iModelViewProjectionMatrix = -1;

    this.Use = function () {

        gl.useProgram(this.prog);

    }

}

// Constructor

function StereoCamera(

    Convergence,

    EyeSeparation,

    AspectRatio,

    FOV,

    NearClippingDistance,

    FarClippingDistance

) {

    this.mConvergence = Convergence;

    this.mEyeSeparation = EyeSeparation;

    this.mAspectRatio = AspectRatio;

    this.mFOV = FOV

    this.mNearClippingDistance = NearClippingDistance;

    this.mFarClippingDistance = FarClippingDistance;

    this.mProjectionMatrix;

    this.mModelViewMatrix;

    this.ApplyLeftFrustum = function (g, viewM) {

        let top, bottom, left, right;

        top = this.mNearClippingDistance \* Math.tan(this.mFOV / 2);

        bottom = -top;

        const a = this.mAspectRatio \* Math.tan(this.mFOV / 2) \* this.mConvergence;

        const b = a - this.mEyeSeparation / 2;

        const c = a + this.mEyeSeparation / 2;

        left = -b \* this.mNearClippingDistance / this.mConvergence;

        right = c \* this.mNearClippingDistance / this.mConvergence;

        this.mProjectionMatrix = m4.frustum(left, right, bottom, top,

            this.mNearClippingDistance, this.mFarClippingDistance)

        this.mModelViewMatrix = m4.identity()

        m4.multiply(m4.translation(0.01 \* this.mEyeSeparation / 2, 0.0, 0.0), this.mModelViewMatrix, this.mModelViewMatrix);

        let modelProjectionMatrix = m4.multiply(this.mProjectionMatrix, m4.multiply(this.mModelViewMatrix, viewM));

        g.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, modelProjectionMatrix)

        g.colorMask(true, false, false, false);

    }

    this.ApplyRightFrustum = function (g, viewM) {

        let top, bottom, left, right;

        top = this.mNearClippingDistance \* Math.tan(this.mFOV / 2);

        bottom = -top;

        const a = this.mAspectRatio \* Math.tan(this.mFOV / 2) \* this.mConvergence;

        const b = a - this.mEyeSeparation / 2;

        const c = a + this.mEyeSeparation / 2;

        left = -c \* this.mNearClippingDistance / this.mConvergence;

        right = b \* this.mNearClippingDistance / this.mConvergence;

        this.mProjectionMatrix = m4.frustum(left, right, bottom, top,

            this.mNearClippingDistance, this.mFarClippingDistance)

        this.mModelViewMatrix = m4.identity()

        m4.multiply(m4.translation(-0.01 \* this.mEyeSeparation / 2, 0.0, 0.0), this.mModelViewMatrix, this.mModelViewMatrix);

        let modelProjectionMatrix = m4.multiply(this.mProjectionMatrix, m4.multiply(this.mModelViewMatrix, viewM));

        g.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, modelProjectionMatrix)

        g.colorMask(false, true, true, false);

    }

}

/\* Draws a colored cube, along with a set of coordinate axes.

 \* (Note that the use of the above drawPrimitive function is not an efficient

 \* way to draw with WebGL.  Here, the geometry is so simple that it doesn't matter.)

 \*/

function draw() {

    gl.clearColor(0, 0, 0, 1);

    gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

    // Set the values of the projection transformation

    let frustrum = 2

    let projection = m4.orthographic(-frustrum, frustrum, -frustrum, frustrum, -frustrum, frustrum);

    // Get the view matrix from the SimpleRotator object.

    let modelView = spaceball.getViewMatrix();

    let rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0.7);

    let translateToPointZero = m4.translation(0, 0, -5);

    let matAccum0 = m4.multiply(rotateToPointZero, modelView);

    let matAccum1 = m4.multiply(translateToPointZero, matAccum0);

    /\* Multiply the projection matrix times the modelview matrix to give the

       combined transformation matrix, and send that to the shader program. \*/

    let modelViewProjection = m4.multiply(projection, matAccum1);

    gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, m4.identity());

    gl.bindTexture(gl.TEXTURE\_2D, camTex);

    gl.texImage2D(

        gl.TEXTURE\_2D,

        0,

        gl.RGBA,

        gl.RGBA,

        gl.UNSIGNED\_BYTE,

        webCam

    );

    camSur.Draw(); gl.clear(gl.DEPTH\_BUFFER\_BIT);

    gl.bindTexture(gl.TEXTURE\_2D, tex);

    const now = Date.now() \* 0.0005;

    gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, m4.translation(0.5 \* Math.sin(now), 0.5 \* Math.cos(now), 0));

    if (audioPos) {

        audioPos.setPosition(0.5 \* Math.sin(now), 0.5 \* Math.cos(now), 0)

    }

    ball.Draw(); gl.clear(gl.DEPTH\_BUFFER\_BIT);

    // Draw the six faces of a cube, with different colors.

    gl.uniform4fv(shProgram.iColor, [1, 1, 0, 1]);

    let dC = document.getElementById('color').value

    let sC = document.getElementById('colorS').value

    gl.uniform3fv(shProgram.iDiffuseColor, [-100, 0, 0]);

    gl.uniform3fv(shProgram.iSpecularColor, hexToRgb(sC));

    gl.uniform2fv(shProgram.iTexball, texball);

    gl.uniform1f(shProgram.iScaler, document.getElementById('scaler').value);

    frustum.ApplyLeftFrustum(gl, matAccum1)

    surface.Draw();

    gl.clear(gl.DEPTH\_BUFFER\_BIT);

    frustum.ApplyRightFrustum(gl, matAccum1)

    surface.Draw();

    gl.colorMask(true, true, true, true);

}

function hexToRgb(hex) {

    var result = /^#?([a-f\d]{2})([a-f\d]{2})([a-f\d]{2})$/i.exec(hex);

    return [

        parseInt(result[1], 16) / 255,

        parseInt(result[2], 16) / 255,

        parseInt(result[3], 16) / 255

    ]

}

function LoadTexture() {

    let texture = gl.createTexture();

    gl.bindTexture(gl.TEXTURE\_2D, texture);

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.LINEAR);

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MAG\_FILTER, gl.LINEAR);

    const image = new Image();

    image.crossOrigin = 'anonymus';

    image.src = "https://raw.githubusercontent.com/Smit0001/VGGI/CGW/texture.jpg";

    image.onload = () => {

        gl.bindTexture(gl.TEXTURE\_2D, texture);

        gl.texImage2D(

            gl.TEXTURE\_2D,

            0,

            gl.RGBA,

            gl.RGBA,

            gl.UNSIGNED\_BYTE,

            image

        );

        console.log("imageLoaded")

        draw()

    }

    return texture

}

function CreateSurfaceData() {

    let vertexList = [];

    let normalList = [];

    let textureList = [];

    let step = parseFloat(document.getElementById('step').value)

    for (let i = 0; i < 1; i += step) {

        for (let j = 0; j < 360; j += 5) {

            vertexList.push(...CreateVertex(i, deg2rad(j)));

            vertexList.push(...CreateVertex(i + step, deg2rad(j)));

            vertexList.push(...CreateVertex(i, deg2rad(j + 5)));

            vertexList.push(...CreateVertex(i, deg2rad(j + 5)));

            vertexList.push(...CreateVertex(i + step, deg2rad(j)));

            vertexList.push(...CreateVertex(i + step, deg2rad(j + 5)));

            normalList.push(...facetAverage(i, deg2rad(j), step));

            normalList.push(...facetAverage(i + step, deg2rad(j), step));

            normalList.push(...facetAverage(i, deg2rad(j + 5), step));

            normalList.push(...facetAverage(i, deg2rad(j + 5), step));

            normalList.push(...facetAverage(i + step, deg2rad(j), step));

            normalList.push(...facetAverage(i + step, deg2rad(j + 5), step));

            textureList.push(i, j / 360);

            textureList.push(i + step, j / 360);

            textureList.push(i, (j + 5) / 360)

            textureList.push(i, (j + 5) / 360)

            textureList.push(i + step, j / 360)

            textureList.push(i + step, (j + 5) / 360)

        }

    }

    return [vertexList, normalList, textureList];

}

function facetAverage(r, b, step) {

    let v0 = CreateVertex(r, b);

    let v1 = CreateVertex(r + step, b);

    let v2 = CreateVertex(r, b + 5);

    let v3 = CreateVertex(r - step, b + 5);

    let v4 = CreateVertex(r - step, b);

    let v5 = CreateVertex(r - step, b - 5);

    let v6 = CreateVertex(r, b - 5);

    let v01 = m4.subtractVectors(v1, v0)

    let v02 = m4.subtractVectors(v2, v0)

    let v03 = m4.subtractVectors(v3, v0)

    let v04 = m4.subtractVectors(v4, v0)

    let v05 = m4.subtractVectors(v5, v0)

    let v06 = m4.subtractVectors(v6, v0)

    let n1 = m4.normalize(m4.cross(v01, v02))

    let n2 = m4.normalize(m4.cross(v02, v03))

    let n3 = m4.normalize(m4.cross(v03, v04))

    let n4 = m4.normalize(m4.cross(v04, v05))

    let n5 = m4.normalize(m4.cross(v05, v06))

    let n6 = m4.normalize(m4.cross(v06, v01))

    let n = calculateAverage([n1, n2, n3, n4, n5, n6])

    n = m4.normalize(n);

    return n;

}

function calculateAverage(arr) {

    let avg = [0, 0, 0]

    for (let i = 0; i < arr.length; i++) {

        for (let j = 0; j < 3; j++) {

            avg[j] += arr[i][j]

        }

    }

    for (let j = 0; j < 3; j++) {

        avg[j] /= arr.length

    }

    return avg;

}

function update() {

    surface.BufferData(...CreateSurfaceData());

    draw();

}

function updateBackground() {

    draw()

    window.requestAnimationFrame(updateBackground)

}

const a = 0.1, n = 1, R = 0.1

function CreateVertex(r, b) {

    const x = r \* Math.cos(b),

        y = r \* Math.sin(b),

        z = a \* Math.cos(n \* Math.PI \* r / R)

    return [x, y, z]

}

let webCam, camTex, camSur;

// Initialize the WebGL context. Called from init()

function initGL() {

    let prog = createProgram(gl, vertexShaderSource, fragmentShaderSource);

    shProgram = new ShaderProgram('Basic', prog);

    shProgram.Use();

    shProgram.iAttribVertex = gl.getAttribLocation(prog, "vertex");

    shProgram.iAttribNormal = gl.getAttribLocation(prog, "normal");

    shProgram.iAttribTexture = gl.getAttribLocation(prog, "texture");

    shProgram.iModelViewProjectionMatrix = gl.getUniformLocation(prog, "ModelViewProjectionMatrix");

    shProgram.iColor = gl.getUniformLocation(prog, "color");

    shProgram.iDiffuseColor = gl.getUniformLocation(prog, "diffuseColor");

    shProgram.iSpecularColor = gl.getUniformLocation(prog, "specularColor");

    shProgram.iTexball = gl.getUniformLocation(prog, "texball");

    shProgram.iScaler = gl.getUniformLocation(prog, "scaler");

    webCam = readCamera()

    frustum = new StereoCamera(0.1, 0.01, 1, 1, 1, 20)

    // add frustum control directly to html

    addFrustumControl(frustum,

        { name: 'Convergence', value: 0.1, min: 0.01, max: 10, step: 0.01 },

        { name: 'Eye Separation', value: 0.01, min: 0, max: 0.75, step: 0.01 },

        { name: 'Field Of View', value: 1, min: 0, max: 2, step: 0.1 },

        { name: 'Near Clipping Distance', value: 1, min: 0, max: 10, step: 0.1 },

    )

    surface = new Model('Surface');

    surface.BufferData(...CreateSurfaceData());

    camSur = new Model('Web cam');

    camSur.BufferData(

        [-1, -1, 0, 1, 1, 0, 1, -1, 0, 1, 1, 0, -1, -1, 0, -1, 1, 0],

        [-1, -1, 0, 1, 1, 0, 1, -1, 0, 1, 1, 0, -1, -1, 0, -1, 1, 0],

        [1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0]);

    let bd = CreateSphereData()

    ball = new Model()

    ball.BufferData(bd, bd, new Array(bd.length).fill(0.8))

    camTex = CreateCameraTexture()

    gl.enable(gl.DEPTH\_TEST);

}

function CreateSphereData() {

    let vertexList = [];

    let u = 0,

        t = 0;

    while (u < Math.PI \* 2) {

        while (t < Math.PI) {

            let v = getSphereVertex(u, t);

            let vv = getSphereVertex(u + 0.1, t);

            let vvv = getSphereVertex(u, t + 0.1);

            let vvvv = getSphereVertex(u + 0.1, t + 0.1);

            vertexList.push(...v);

            vertexList.push(...vv);

            vertexList.push(...vvv);

            vertexList.push(...vvv);

            vertexList.push(...vv);

            vertexList.push(...vvvv);

            t += 0.1;

        }

        t = 0;

        u += 0.1;

    }

    return vertexList;

}

function getSphereVertex(long, lat) {

    return [

        Math.cos(long) \* Math.sin(lat) \* 0.1,

        Math.sin(long) \* Math.sin(lat) \* 0.1,

        Math.cos(lat) \* 0.1

    ]

}

/\* Creates a program for use in the WebGL context gl, and returns the

 \* identifier for that program.  If an error occurs while compiling or

 \* linking the program, an exception of type Error is thrown.  The error

 \* string contains the compilation or linking error.  If no error occurs,

 \* the program identifier is the return value of the function.

 \* The second and third parameters are strings that contain the

 \* source code for the vertex shader and for the fragment shader.

 \*/

function createProgram(gl, vShader, fShader) {

    let vsh = gl.createShader(gl.VERTEX\_SHADER);

    gl.shaderSource(vsh, vShader);

    gl.compileShader(vsh);

    if (!gl.getShaderParameter(vsh, gl.COMPILE\_STATUS)) {

        throw new Error("Error in vertex shader:  " + gl.getShaderInfoLog(vsh));

    }

    let fsh = gl.createShader(gl.FRAGMENT\_SHADER);

    gl.shaderSource(fsh, fShader);

    gl.compileShader(fsh);

    if (!gl.getShaderParameter(fsh, gl.COMPILE\_STATUS)) {

        throw new Error("Error in fragment shader:  " + gl.getShaderInfoLog(fsh));

    }

    let prog = gl.createProgram();

    gl.attachShader(prog, vsh);

    gl.attachShader(prog, fsh);

    gl.linkProgram(prog);

    if (!gl.getProgramParameter(prog, gl.LINK\_STATUS)) {

        throw new Error("Link error in program:  " + gl.getProgramInfoLog(prog));

    }

    return prog;

}

//initialization function that will be called when the page has loaded

let tex;

function init() {

    let canvas;

    try {

        canvas = document.getElementById("webglcanvas");

        gl = canvas.getContext("webgl");

        tex = LoadTexture()

        if (!gl) {

            throw "Browser does not support WebGL";

        }

    }

    catch (e) {

        document.getElementById("canvas-holder").innerHTML =

            "<p>Sorry, could not get a WebGL graphics context.</p>";

        return;

    }

    try {

        initGL();  // initialize the WebGL graphics context

    }

    catch (e) {

        document.getElementById("canvas-holder").innerHTML =

            "<p>Sorry, could not initialize the WebGL graphics context: " + e + "</p>";

        return;

    }

    spaceball = new TrackballRotator(canvas, draw, 0);

    draw();

    updateBackground()

}

function readCamera() {

    const video = document.createElement('video');

    video.setAttribute('autoplay', true);

    navigator.getUserMedia({ video: true, audio: false }, function (stream) {

        video.srcObject = stream;

    }, function (e) {

        console.error('Rejected!', e);

    });

    return video;

}

function CreateCameraTexture() {

    const cameraTexture = gl.createTexture();

    gl.bindTexture(gl.TEXTURE\_2D, cameraTexture);

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.LINEAR);

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MAG\_FILTER, gl.LINEAR);

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_S, gl.CLAMP\_TO\_EDGE);

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_T, gl.CLAMP\_TO\_EDGE);

    return cameraTexture;

}

let audioContext;

let audio = null;

let src,

    lowpassFilter,

    audioPos;

function setupAudioListerners() {

    audio = document.getElementById('audioElement');

    audio.addEventListener('play', () => {

        if (!audioContext) {

            audioContext = new AudioContext();

            src = audioContext.createMediaElementSource(audio);

            audioPos = audioContext.createPanner();

            lowpassFilter = audioContext.createBiquadFilter();

            src.connect(audioPos);

            audioPos.connect(lowpassFilter);

            lowpassFilter.connect(audioContext.destination);

            lowpassFilter.type = 'lowpass';

            lowpassFilter.Q.value = 0.66;

            lowpassFilter.frequency.value = 66;

            audioContext.resume();

        }

    })

    audio.addEventListener('pause', () => {

        console.log('pause');

        audioContext.resume();

    })

}

function initAudio() {

    setupAudioListerners();

    let filterListener = document.getElementById('filterEnabled');

    filterListener.addEventListener('change', function () {

        if (filterListener.checked) {

            audioPos.disconnect();

            audioPos.connect(lowpassFilter);

            lowpassFilter.connect(audioContext.destination);

        } else {

            audioPos.disconnect();

            audioPos.connect(audioContext.destination);

        }

    });

    audio.play();

}